2

- *Diyalogların oluşturulması*
- *Detaylarıyla Sinyal ve dilimler*
- *Hızlı diyalog tasarımı*
- *Şekil deşiştiren diyaloglar*
- *Mütaharrik diyaloglar*

Diyalogların Oluşturulması

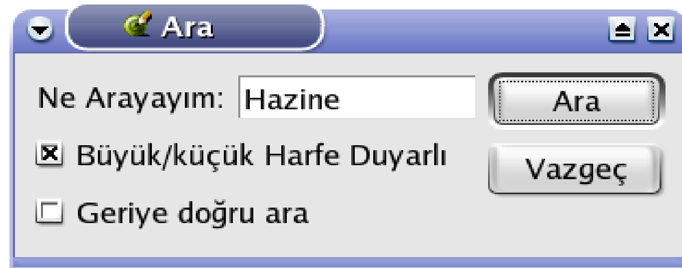
Bu bölümde Qt kullanarak nasıl *diylog kutuları* olüşturulabileceğini öğreneceksiniz. Onların *diyalog kutuları* (bazan sadece *diyaloglar*) diye adlandırılmalarının sebebi kullanıcı ile proğram arasında diyaloğu sağlamalrıdır.

Diyaloglar kullanıcıya seçenekler ve ayarlar takdim ederler ve kullanıcının arzu ettiği şekilde seçenekleri ayarlamasına imkan sağlarlar. Çoğu GUI (grafik kullanıcı arabirimi) proğramları bir ana pencere, menü çubuğu, araç çubuğu ile ana pencereyi tamamlayan onlarca diyalogtan mütekevvindirler. Sadece bir veya daha fazşa diyalogdam muteşekkil, kullanıcı komutlarına kariılık veren proğramlar (hesap makinesi gibi) yazmakta mümkündür.

İlk diyaloğu kaynak kodunun tamamını, sizin görmüş olmanız için, yazmak suretiyle oluşturacağız. Sonra bunun *Qt Designer* programında daha kolay bir şekilde nasıl yapılabileceğini izah edeceğiz. Qt Desıgner kullanarak proğram yazmak kaynak kodunun tamamını yazmaktan çok daha hızlı olduğu gibi değişik tasarımları kolayca deneme imkanıda sağlar.

QDialog dan bir altsınıf Oluşturulması

İlk misalimiz tamamen C++ ile yazılmış Ara diyaloğu olacak. Biz bunu, diyaloğu başlı başına bir sınıf yapmak suretiyle gerçekleştireceğiz. Bu durumda o bağımsız, kendi içinde bir bütün oluşturan ve kendine has *sinyal* ve *dilimleri* olan bir eleman olmuş olur.



Şekil 2.1: Ara diyaloğu (Red Hat Linux KDE)

Kaynak kodu `finddialog.h` ve `finddialog.cpp` adındaki iki ayrı dosyada yer almaktadır. Tabii olarak `finddialog.h` ile başlayacağız.

```
1 #ifndef FINDDIALOG_H
2 #define FINDDIALOG_H
3 #include <qdialog.h>
4 class QCheckBox;
5 class QLabel;
6 class QLineEdit;
7 class QPushButton;
```

Satır 1 ve 2 (ve 27) başlık dosyalarının bir defadan fazla dahil edilmelerini önler.

Satır 3, Qt deki bütün diyalogların temelini teşkil eden `QDialog` sınıfının tanımını dahil eder. `QDialog`, `QWidget` in alt sınıfıdr.

Satır 4 ten 7 ye kadar daha sonra kullanılcak sınıfların *ileri bildirimi* (forward declaration) yapılır. İleri bildirim C++ derleyicisine bu sınıfların varlığını ihbar eder ancak bu sınıfların detayları hakkında, ki onalr genelde sınıfın kendi başlık dosyasında bulunur, herhangi bir bilgi vermez. Birazdan bu konuya tekrar değineceğiz. Şimdi `FindDialog` namında `QDialog` un alt sınıfı olan bır sınıf oluşturalım:

```
 8 class FindDialog : public QDialog
 9 {
10    Q_OBJECT
11 public:
12    FindDialog(QWidget *parent = 0, const char *name = 0);
```

Sinyal (`signal`) ve dilimleri (`slot`) olan her sınıfın başında `Q_OBJECT` makrosu bulunmak zorundaır. `FindDialog` sınıfının yapıcısı (satır 12) Qt sınıflarında kullanılan tipik bir yapıcıdır. Bağımsız değişkenlerden olan `parent` ebeveyn widgeti belirtirken `name` ismini belirtir. İsim parametresinin kullanılması ihtiyaridir, genelde hata bulma ve proğram deneme maksadıyla kullanılır.

```
13 signals:
14    void findNext(const QString &str, bool caseSensitive[1]);
15    void findPrev(const QString &str, bool caseSensitive);
```

Satır 13 deki `signals` bölümünün altında (14. ve 15. satırlarda) kullanıcının *Ara* düğmesine tıkladığı anda yayınlanabilecek iki sinyal tanımlanmaktadır. Eğer kullanıcı geriye doğru ara seçeneğini seçmiş ise `findPrev()`[2], aksi takdirde `findNext()`[3] sinyali yayınlanır.

Bir anahtar sözcük olan `signals` aslında bir makrodur. C++ ön işlemcisi onu derleyici görmeden normal C++ koduna çevirmektedir.

```
16 private slots:
17    void findClicked();
18    void enableFindButton(const QString &text);
19 private:
20    QLabel *label;
```

[1] Büyük/küçük harfe duyarlı

[2] findPrev(): öncekini bul

[3] findNext(): bir sonrakini bul

```
    QLineEdit *lineEdit;
    QCheckBox *caseCheckBox;
    QCheckBox *backwardCheckBox;
    QPushButton *findButton;
    QPushButton *closeButton;
};
#endif
```

Sınıfın `private` bölümünde iki tane dilim (`slots`) tanımladık. Bu dilimleri kullanabilmemiz için dıaloğun çocıklarından çoğusuna ulaşabilmeliyiz, bunun için onlrın herbirine yönelik işaretçişere (*pointers*) ihtiyacımız olacak. `Slots` anahtar sözcüğü, `signals` gibi, bir makro oluğ C++ ön işlemcisi tarafından açılmaktadır taki derletici onu hazmedebilsin.

Buradaki bütün değişkenler *işaretçi* olduklarından be başlik dosyası (`finddialog.h`) içerisinde bizzat kullanılmadıklarından, derleyicin onlar hakkında detaylı bilgiye sahıp olması gerekmez ve bu aşamada ileri bildirim kafidir. Biz istesydik müsait başlık dosyalarını (`<qcheckbox.h>`, `<qlabel.h>`, vesaire) dahil edebilirdik, ancak mümkün ise ileri bildirimin kullanılması derleme işlemini azda olsa hızlandırır.

Şimdi `FindDialog` sınıfının kaynak kodunu içeren `finddialog.cpp` dosyasına bakalım.

```
1 #include <qcheckbox.h>
2 #include <qlabel.h>
3 #include <qlayout.h>
4 #include <qlineedit.h>
5 #include <qpushbutton.h>
6 #include "finddialog.h"
```

Önce, finddialog.h dosyasına ilaveten Qt sınıflarından kullanacak olduğumuz sınıfların başlık dosyalarını dahil ediyoruz. Qt sınıflarının çoğunun başlık dosyaşarı sınıfın ismi ile aynı adı taşır ancak bütün harfler küçük yazılır ve `.h` uzantısını alı.

```
 7 FindDialog::FindDialog(QWidget *parent, const char *name)
 8 : QDialog(parent, name)
 9 {
10    setCaption(trUtf8("Ara"));
11    label = new Qlabel(trUtf8("cNe arayay m:"), this);
12    lineEdit = new QLineEdit(this);
13    label->setBuddy(lineEdit);
14    caseCheckBox = new QcheckBox(trUtf8("Büyük/küçük harfe &duyarl "),
this);
15    backwardCheckBox = new QcheckBox(trUtf8(&Geriye do ru  ara"),
this);
16    findButton = new QPushButton(trUtf8("&Ara"), this);
17    findButton->setDefault(true);
18    findButton->setEnabled(false);
19    closeButton = new QPushButton(trUtf8("&Vazgeç"), this);
```

Satı 8 de, parent ve isim parametrelerini temel sınıfın yapıcısına gönderiyoruz.

Satır 10 da pencernin başlığını `Ara` yapıyoruz. Yine aynı satırdakı `trUtf8()`[4]içerisinde yer alan metin kolyca başka lisanlara çevrilebilir. Bu fonksiyon QObject sınıfı içerisinde tanımlanmış olup Q_OBJECT macrosunu içeren her sınıf içerisinde de mevcuttur. Her ne kadar proğramınızı başka dillere çevirmeyi düşünmüyür isenizde kullanıcının görmesi ihtimal olan metinleri `trUtf8()` içerisine almanız önemle tavsiye olunur. Qt proğramlarının başka lisanlara nasıl çevrilebilecekleri 15. bölümde anlatılacaktır.

Sonra çocuk widgetları oluşturacagız. & işareti *hızlandırılmış tuşlar* için kullanılır. Mesela, 16. satırda bir Ara düğmesi oluşturulmaktadır ki kullanıcı ona Alt+A tuşlarını kullanarak ulaşılabilinir. Aynı işaret odak ayarı içinde kullanılır: Satır 11 de, hızlandırılmış tuşu Alt+N olan bir etiket (label) oluşturduk. bir etiket oluşturduk ve 13. satırda bir sati editörünü ona *ahbap* (buddy) yaptık. Burada ahbap dan maksay bir widgetin hızlandırılmış tuşlarına basıldığında onun ahbabı olan widget odak noktası olur. Yani Alt+N (etiketin hızlandırılmış tuşları) tuşlarına basıldığında odak noktası satır editörü (ahbap) olur.

Satır 17 de [5] fonksiyonunu çağırmak suretşyle Ara düğmesini varsayılan düğme halin getirdik. Varsayılan düğme enter tuşuna basıldığında tıklanan düğmedir. Satır 18 de Ara düğmesini pasif hale getirdik, bu durumda bu düğme kullanıcı komutlarına karşılık vermez ve soluk renk alır.

```
20    connect(lineEdit, SIGNAL(textChanged(const QString &)),
21          this, SLOT(enableFindButton(const QString &)));
22    connect(findButton, SIGNAL(clicked()),
23          this, SLOT(findClicked()));
24    connect(closeButton, SIGNAL(clicked()),
25          this, SLOT(close()));
```

Özel (private) bir dilim (slot) olan `enableFindButton(const QString &)` ne zamanki satır editöründeki metin değişince hemen çağrılır. Kullanıcı Ara diğmesine tıklayınca özel dilim `findClicked()` koşturulur. Vazgeç düğmesine tıklandığında diyalog kendisini kapatır. Close() dilimi QWidget dan alınmıştır, ve normalde widgeti saklar. Birazdan `enableFindButton()` and `findClicked()` dilimlerinin kaynak koduna göz atacağız. QObject, FindDialog sınıfının ebeveynlerinden birisi olduğu için connect() in önünde QObject:: önekini kullanmamıza gerek yoktur.

```
26    QHBoxLayout *topLeftLayout = new QHBoxLayout;
27    topLeftLayout->addWidget(label);
28    topLeftLayout->addWidget(lineEdit);
29    QVBoxLayout *leftLayout = new QVBoxLayout;
30    leftLayout->addLayout(topLeftLayout);
31    leftLayout->addWidget(caseCheckBox);
```

---

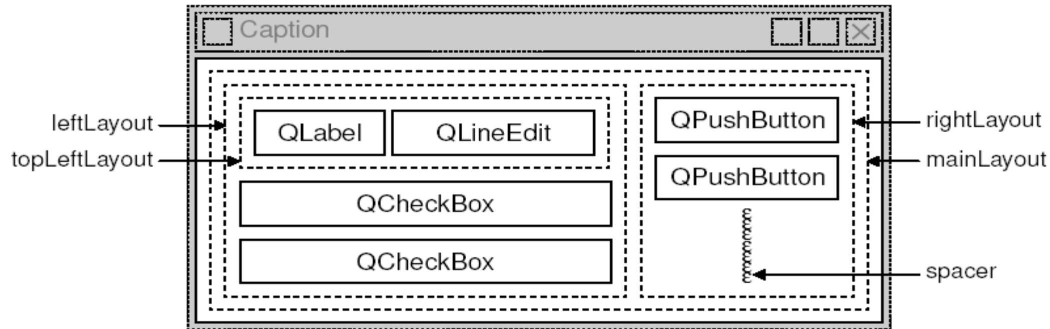
[4] Şayet metin ACII harflerinin haricinde olan ş,ı,ğ gibi Türkçemize veya başka lisanlara mahsus hafler içermiyor ise o takdirde tr() fonksiyonunuda kullanabilirsiniz.

[5] Qt `TRUE` and `FALSE` anahtar sözcüklerini bütün işletim sistemleri için tanımlamaktadır, ancak yeni derleyicilerin çoğu `true` ve `false` tanımlamaktadır. Bazı eski derleyiciler `true` ve `false` tanımıyor olabilirler böyle bir durumda Qt nin tanımladığı `TRUE` ve `FALSE` kullanılabilir aksi takdirde standard C++ da mevcut olan `ture` ve `false` kullanmanız tavsiye olunur.

```
    leftLayout->addWidget(backwardCheckBox);
    QVBoxLayout *rightLayout = new QVBoxLayout;
    rightLayout->addWidget(findButton);
    rightLayout->addWidget(closeButton);
    rightLayout->addStretch(1);
    QHBoxLayout *mainLayout = new QHBoxLayout(this);
    mainLayout->setMargin(11);
    mainLayout->setSpacing(6);
    mainLayout->addLayout(leftLayout);
    mainLayout->addLayout(rightLayout);
}
```

Nihayet *düzen* (layout) *mekanızmalarını* kullanarak çocuk widgetları düzene soktuk. Düzen mekanizması (layout manager) bir nesen olup widgetların ebat ve mekanlarını ayarlar. Qt de mevcut 3 düzen mekanizması vardır: `QHBoxLayout` widgetları yatay olarak soldan sağa doğru düzenelr, `QVBoxLayout` widgetları düşey olarak yukarıdan aşağıya doğru düzenler ve `QGridLayout` ise klavuz (izgara) şeklinde düzenler.

Düzenler, widgetlari ihtiva ettikleri gibi diğer düzenleride içerebilirler. `QHBoxLayout`, `QVBoxLayout` ve `QgridLayout` düzen mekanizmalarını mütedahil etmek suretiyle kompleks diyaloglar oluşturmak mümkündür.



Şekil 2.2: Ara diyaloğunun düzenleri

Şekil 2.2 de ibraz ve teşhir edildiği gibi Ara diyaloğunun hazırlanmasında iki tane `QHBoxLayout` ve iki tanede `QVBoxLayout` istımal edildi. En dışta yer alan dizim diyaloğun ana dizimidir (`mainLayout`); FindDialog nesnesi (`this`) bu dizimin ebeveyni olmakla birlikte onun düzeni ana dizimin sorumluluğundadır. Diğer dizimleri üçü Geri kalan diğer üç dizim ise yardımcı dizimlerdir. Şekil 2.2 de sağ alt trafta kalan yay boşluk[6] bırakmak için kullanılır. Bu *aralıkçı* (`spacer`) düğmelerin altında kalan boşluğu doldurmak suretiyle *Ara* ve *Vazgeç* düğmelerinın sağ dizimin (`rightLayout`) üst kısmında yer almalarını sağlar.

6 `spacer` aralık, boşluk bırakan fasıla veren demektir.

Dizim mekanizmaları hakkında akıldan çıkarılmaması gereken bir husus bu sınıfların widget olmamalarıfır. Bilakis, onların herbiri `QLayout` sınıfının birer varisleri olup `QLayout` un kendiside `QObject` sınıfının bir varisidir. Yukarıdaki şekilde widgetlar ın etrafları sürekli çizgi ile çevrilmiş olup dizimler ise kesikli çizgi ile gösterilmektedirle. Normal bir proğramda dizimler görünmezler.

Dizim mekanizmaları widget olmadıkları halde hem *ebeveyn* sahibi hemde *çocuk* sahibi olabilirler. Dizimler için ebeveyn in anlamı wıdgetlardakinden biraz farklıdır. Ebeveyni widget olan bir dizim inşa edildiğinde (yukarıdaki `mainLayout` da olduğu gibi), layout o widgetın yüzeyini kaplar. Şayet bir dizim, yukarıdaki `topLeftLayout`, `leftLayout`,and `rightLayout` misallerinde olduğu gibi, ebevynsiz oluşturulursa bu dizim daha sonra `addLayout()` fonksiyonu kullanılarak bir başka dizime dahil edilmelidir.

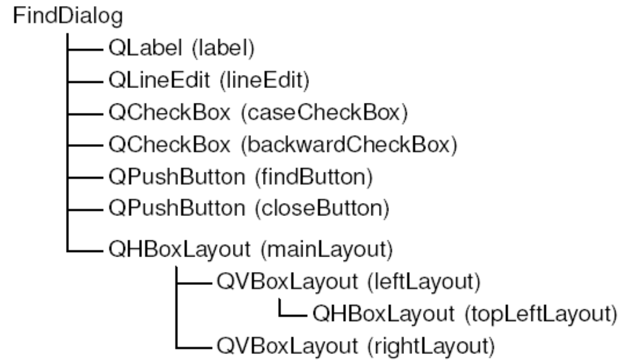
Qt nin ata (ebeveyn) çocuk ilişkisi QWidget ve QLayout sınıflarının temelini teşkil eden QObject sınıfnında yer almaktadır. Bir ata ya sahip olan nesne oluşturduğumuzda, ister bir widget ister bir dizm veya harhangi başka bir çeşit, ata bu yeni oluşturulan nesneyi onun çocukları listesine zammeder. Ata silindiği zaman o önce kendi çocukları listesindeki bütün nesneleri siler. Bu esnada herbir çocukta, şayet varsa, kendi çocuklarının listesinde bulunan bütün nesneleri siler be bu iş bütün çocuklar silininceye kadar mütevaliyen ve mütetabiyen devam eder.

Ata çocuk mekanizması *hafıza* (bellek) idaresini son derece kolaylaştırıp hafıza sızıntısını mimkin mertebe azaltır. Proğramcının bizzat silmesi gereken nesneler atası olmayan ve *new* ile oluşturulmuş nesnelerdir. Şayet bir çocuğu atasından önce silecek olursanız Qt bu nesneyi otomatik olarak atasının çocukları listesinden çıkarır ve sile

Widgetlar için atalık (ebeveynlik) biraz farklıdır: Çocuk widgetlar atalarının içerisinde teşhir edilirler. Ata silinir silinmez çocuklar hafızadan silindikleri gibi ekrandanda kaybolurlar.

Bir dizimi bir bşka dizime `addLayout()` fonksüyonununu ile dahil ettiğimizde dahil edilen dahil olunanın çocuğu olur buda hafıza idaresini teshil eder. Dizimlerin aksine, bir widgeti bir dizime ilave ettiğiniz takdirde o widgetın atası değişmez.

Şekil 2.3 widget ve dizimlerin nesil ve soylarını izhar etmektedir. Nesil ve soylar FındDialog sınıfının yapıcısının kodundakı new ve addLayout() ihtıva eden satırlara bakmak suretiyle istidlal edilebilir. Akılda tutlması gereken bir nokta şudurki dizimler tanzim ettikleri widgetların ataları degillerdir.



Şekil 2.3: Ara diyaloğunun ata çocuk ilişkisi

Dizim mekanizmalarına ilaveten, Qt birtakım *dizim widgetları* da tedarik etmektedir: `QHBox` (birinci bölümde kullandık), `QVBox` ve `QGrid`. Bu sınıflar çocukları olan widgetlar için hem *ata* vazifesi hemde *dizim mekanizması* vazifesi görürler. Küçük misaller için *dizim widgetlarını* kullanmak *dizim mekanizmalarını* kullanmaktan daha elverişlidir ancak onlar mekanizmalar kadar uysal olmadıkları gibi daha fazla imkanları (hafıza gibi) gerektirirler.

Böylece `FindDialog` sınıfının *yapıcısını* (constructor) ğözden geçirmiş olduk. Biz bu diyaloğun widgetlaını ve dizimlerini `new` kullanarak oluşturduğumuz için bütün bunları hafızadan silecek bir *yıkıcıya* (destructor) gerek olduğu düşünülebilir. Ancak buna gerek yoktur çünkü Qt atası silinen bütün nesneleri otomatik olarak siler ve oluiturduğumuz nesnelerin hepsi `FindDialog` un neslindendirler (yani ya çocuğu yada torunu vesaire.)

Şimdi diyaloğun *dilimlerini* (`slots`) gözden geğireceğiz:

```
void FindDialog::findClicked()
{
   QString text = lineEdit->text();
   bool caseSensitive = caseCheckBox->isOn();
   if (backwardCheckBox->isOn())
         emit findPrev(text, caseSensitive);
   else
         emit findNext(text, caseSensitive);
}
void FindDialog::enableFindButton(const QString &text)
{
   findButton->setEnabled(!text.isEmpty());
}
```

Kullanıcı *Ara* düğmesine tıkladığında `findClicked()` *dilimi* çağrılır. *Geriye doğru ara* seçeneğine bağlı olarak ya `findPrev()` yada `findNext()` *sinyalini* yayınlar. Bir anahtar sözcük olan emit (yayınla) Qt ye mahsus olup C++ önişlemcisi tarafından standart C++ diline çevrilir.

Ne zamanki kullanıcı *satır editörindeki* metni değiştirirse *enableFindButton()* dilimi çağrılır. *Satır editöründe* bir metin mevcut olduğu müddetçe *Ara* düğmesini *muktedir* aksi taktirde *malul* yapar.

Bu iki dilim ile diyalog tekmil edilmiş oldu. Şimdi artık her C++ proğramı için gerekli olamn main() fonksiyonunu main.cpp dosyasına kaydedip FindDialog widgetımızı deneyedebiliriz:

```
#include <qapplication.h>
#include "finddialog.h"
int main(int argc, char *argv[])
{
   QApplication app(argc, argv);
   FindDialog *dialog = new FindDialog;
   app.setMainWidget(dialog);
   dialog->show();
   return app.exec();
}
```

Bu proğramı derlemek için *qmake* proğramını alelade bir şekilde koştur. `FindDialog` sınıfının tanımında `Q_OBJECT` makrosu yer aldığından, *qmake* tarafından *Makefile* içerisine *moc*[7] proğramını (Qt ye has *meta-object derleyicisi*) koşturacak hususi kurallar yerleştirilir.

Moc derleyicisinin doğru çalışabilmesi için sınıfın tanımının bir başlık dosyasında, sinifin kaynak kodundan ayrı bir dosyada, yer alması gerekir. Bu başlık dosyası *moc* tarafından husule getirilen kod içerisine dahil edilir.

Q_OBJECT makrosunu kullanan sınıflar *moc* proğramı ile derlenmelidirler. Bu proğramcı için bir zorluk tesşkil etmez çünkü *qmake* şüzümlu komutları *Makefile* dosyasına otomatşk olarak ekler. Eğer Makefile dosyasını qmake kullanarak yeniden oluşturmayı şhmal ederseniz bu durumda moc koşturulmayacak ve *bağlayıcı* (linker) bir takım fonksiyonların tanımlandığını ancak kaynak kodlarının bulunamadığna dair hata/uyarı mesajı verecektir. Bu mesaj genelde vazih degildir.

GCC aşağıdakine benzer uyarılar verir:

```
finddialog.o(.text+0x28): undefined reference to
FindDialog::QPaintDevice virtual table
```
[8]

Visual C++ s output starts like this:

```
finddialog.obj : error LNK2001: unresolved external symbol
"public:~virtual bool __thiscall FindDialog::qt_property(int,
int,class QVariant *)"
```
[9]

Böyle bir mesajla karıılaitığınızda qmake proğramını koşturup Makefile dosyasını yeniden oluşturp proğramı tekrar derleyiniz.

Şimdi proğramı çaliştiriniz ve hızlandırılmış tuşlar olan `Alt+N`, `Alt+V`, `Alt+G` ve `Alt+A` tuşlarının gerektiği şekilde işleyip işlemediklerini tahkik ediniz. Tab tuşuna basarak widgetlar arasında seyrediniz. *Tab* tuşuna bastığınızda widgetları hangi sırada ziyaret

---

[7] İngilizce **m**eta-**o**bject **c**ompiler ibaresinin kısaltılmışı olup mok/mak diye telaffuz edilir.

[8] finddialog.o(.text+0x28): FindDialog::QPaintDevice hayali listesine yapılan mübhem bir havale.

[9] finddialog.obj : hata LNK2001: halledilemeyen harici sembol "public:~virtual bool __thiscall FindDialog::qt_property(int, int,class QVariant *)"

edeceğiniz, ki buna *tab sırası* adı verilir, onlarin oluşturulma sırasına bağlıdır. Yani *tab sırsaı* ilk oluiturulandan son oluşturulana doğrudur. Ancak bu sıra `Qwidget::setTabOrder()` kullanılarak değiştirilebilir.

Makul bir tab sırasının mevcudiyeti ve yeterince hızlandırılmış tuşların varlığı fare kullanmak istemeyen veya kullanamayan kullanıcının proğrama tam hakimiyetini mümkün kılar. Proğramın kılaviyeden kontrol edilebilmesi hızlı yazan kişiler tarafındanda tercih edilmektedir.

Üçüncü bölümde *Ara* diyaloğunu gerçek bir proğram içersisinde kullanacağız ve `findPrev()` ile `findNext()` sinyallerini geröek dilimlere bağlayacağız.

Detaylarıyla Sinyal ve dilimler

Sinyal ve *dilim* mekanizması Qt nin temel taşlarından birisini teşkil eder. Bu proğramcının nesnelerin, birbirleri hakkında bilgi sahibi olmalarına grek kalmaksızın, diğer nesneler ile rabtedebilmesini teshil eder. Şu ana kadar bir kısım sinyalleri dilimlere bağladık, sinyaller ile dilimler tanımladık, bazı dilimlerin kaynak kodunu tamamen kendimiz yazdık ve bizce oluğturulan sinyaller yayınladık. Şimdi bu mekanizmaya biraz daha yakından bakalım.

Dilimler neredeyse normal C++ fonksüyonlarından farksızdırlar. Onlar *hayali* (virtual) olabilecekleri gibi *aşırı yüklü* (overloaded), *hususi* (`private`), *umumi* (`public`) veya *mahfuz* (`protected`) olabilirler ve sıradan bir C++ *uzvy fonksiyonu* (member function) gibi çağrılabilir. Tek fark *dilimlerin sinyallere* bağlanabilmeleridirki her defasında sözkonusu sinyal yayınlandığında ona bağlı olan dilim koşturulur.

`Connect()` fonksiyonu şu genel şekliyle kullanılır:

```
connect(sender, SIGNAL(signal), receiver, SLOT(slot));
```

burada `sender` (gönderen) ve `receiver` (alıcı) QObject sınıflarının *timsalleri* (pointers) olup `signal` ve `slot` fonksiyon mühürleri olup değişken isimlerini içermezler. `SIGNAL()` ve `SLOT()` makroları aslında argumanlarını metne çevirirler. Şu ana kadar rastladığımız misallerde her defasında muhtelif *sinyal* ve *dilimleri* yekdiğeri ile bağladık. Daha başka ihtimaller de mevcuttur:

- **Bir sinyal birden fazla dilime raptedilebilir:**

```
connect(slider, SIGNAL(valueChanged(int)),
      spinBox, SLOT(setValue(int)));
connect(slider, SIGNAL(valueChanged(int)),
      this, SLOT(updateStatusBarIndicator(int)));
```

  `valueChanged(int)` sinyali yayınlandığında bağlı bulunduğu dilimlerden her biri keyfi bir şekilde çağrılırlar.

- **Birden fazla sinyal tek bir dilime bağlanabilir:**

```
connect(lcd, SIGNAL(overflow()),
      this, SLOT(handleMathError()));
```

```
connect(calculator, SIGNAL(divisionByZero()),
      this, SLOT(handleMathError()));
```

Sinyallerden herhangi birisi yayınlandığında dilim çağrılır.

- **Bir sinyal bir diğer sinyale bağlanabilir:**

```
connect(lineEdit, SIGNAL(textChanged(const QString &)),
      this, SIGNAL(updateRecord(const QString &)));
```

İlk sinyal yayınlandığında ikinciside yayınlanır. Bunun haricinde bir sinyalin diğer bir sinyale raptı ile bir sinyalin bir dilime irtibatı arasında hiç bir fark yoktur.

- **Bağlantılar silinebilir:**

```
disconnect(lcd, SIGNAL(overflow()),
      this, SLOT(handleMathError()));
```

Bu nadiren gereklidir çünkü Qt bir nesne silidiği zaman onunla muteallik olan bğtğn bağlantıları siler.

Bir *sinyalin* bir *dilime* (yahut bir başka *sinyale*) raptedilebilmesi için her ikisininde aynı sayıda, türde ve sırada değişkenleri olmalıdır:

```
connect(ftp, SIGNAL(rawCommandReply(int, const QString &)),
      this, SLOT(processReply(int, const QString &)));
```

İstisnai olarak, şayet bir sinyal bağlanmış olduğu dilimden daha fazla sayıda değişkene sahip ise ziyade değişkenler ihmal edilir:

```
connect(ftp, SIGNAL(rawCommandReply(int, const QString &)),
      this, SLOT(checkErrorCode(int)));
```

Eğer değişken türleri *telifi kabil* (compatible) değilseler veyahut dilim yada sinyalden herhangi birisi yoksa, o takdirde proğram koşturulduğu esnada Qt uyarı mesajı verir. Aynı şekilde değişken isimleri *sinyal* veya *dilim* ile birlikte mevcut ise proğram çalıştırıldığında Qt uyarı mesajı verir.

Qt nin Meta-Nesne Sistemi

Qt nin pek büyük katkilarindan birisi, farklı yazılım (software) unsurlarının yekdiğeri hakkında malumat sahibi olmaksızın birbirlerine bağlanabilmelerini mümkün kılan bir C++ temdididir (extension).

Bu mekanizmaya meta-nesne sistemi (meta-object system) adı verilir ve Qt için büyük ehemmiyet arzeden *sinyal* ve *dilim* mekanizması ile *tedkik-i batıni* (introspection) tedarik eder. *Tedkik-i batıni* sinyal ve dilim mekanizmasının çalışması için gerekli olduğu gibi proğramcının proğram çalışırken QObject alt sınıfları hakkında, sınıfların isimleri, sinyal ve dilim listeleri gibi meta-bilgiler edinmesine imkan sağlar. Bu mekanizma aynı zmanda *Qt Desiğner* proğramındaki *özellikler* (properties) ile *metin tercümesi* (internasyonalizasyon) içinde kullanılır.

Standart C++ dili Qt nin meta-nesne sistemi tarafından elzem olan mutaharrik (dinamik) meta bilgileri tedarik etmemektedir. Qt bu problemi *moc* isminde müstakil bir proğram kullanarak halletmektedir ki bu proğram `Q_OBJECT` sınıf tanımını tahlil edip (parse) onu standart C++ fonksiyonlarına çevirir. *Moc* görevini ifa ederken sadece saf C++ kullandığı için Qt nin meta-nesne sistemi bütün derleyiciler ile telifi kabildir (compatible).

Bu mekanizmanın çalışma şekli kısaca şöyledir:

- `Q_OBJECT` makroları bir takım *tedkik-i batıni* fonksiyonları tanımlamaktadırki bunlar her bir `QObject` alt sınıfalrı içerisinde tanımlanmiş olmalıdırlar: `metaObject(), className(), tr()` vesaire.
- *Moc* proğramı `Q_OBJECT` tarafından tanımlanmış olan her bir fonksiyon ve bütün sinyaller için kaynak kodu oluşturur.
- `QObject` sınıfının *uzvi fonksiyonları* (member functions) olan `connect()` and `disconnect()` fonksiyonları *tedkik-i batıni* fonksiyonlarını ıstimal etmek suretiyle görevlerini yerine getirirler.

Bütün bunlar qmake, moc ve `QObject` tarafından otomatik olarak idare edilir taki sizin bu tür detaylara inmenize çok ender ihtiyaç vardır. Şayet merak ediyorsanız `moc` tarafından husule getirilen C++ kaynak kodunu incelemek suretiyle daha detaylı bilgi edinebilisiniz.

Şu ana kadar, sinyal ve dilimleri sadece widgetlar ile kullandık. Ancak sinyal ve dilim mekanizmasi QObject sınıfında mevcut olduğu için kullnım alanı sadece GUI ile sınırlı değildir. Bu mekanizma QObject sınıfının bütün alt sınıflarında kullanılabilir:

```
class Employee[10] : public QObject
{
      Q_OBJECT
public:
```

[10] İşçi

```
        Employee() { mySalary[11] = 0; }
        int salary[12]() const { return mySalary; }
    public slots:
        void setSalary[13](int newSalary);
    signals:
        void salaryChanged[14](int newSalary);
    private:
    int mySalary; };
    void Employee::setSalary(int newSalary)
    {
        if (newSalary != mySalary) {
            mySalary = newSalary;
            emit salaryChanged(mySalary);
        }
    }
```

`setSalary()` fonksiyonunun nasıl oluşturulduğuna dıkkat edin. Biz `salaryChanged()` *sinyalini* sadece `newSalary` değişkeninin `mySalary` değişkeninden farklı olması durumunda (`if newSalary != mySalary`) yayınlıyoruz. Buda müteselsil bağlantaıların (cyclic connections) *sonsuz döngüye* girmesine mani olur.

Süratli Diyalog Tasarımı

Qt tertib edilirken, proğram yazması hem zevkli hemde kolay olması göz önünde bulundurulmuştur. Bu yüzden saf C++ yazmak suretiyle Qt de proğram oluşturulabilir. Qt Designer proğramcının grafiksel tasarlanmış formları kaynak kodu ile birleştirmesini mümkün kılar.

Bu kısımda *Qt designer* kullanarak şekil 2.4 te teşhir edilen *Hücreye Git* (Go-to-Cell) diyaloğu oluşturacağız. Bir diyaloğun meydana getirilmesi, ister *Qt designer* kullanarak isterse *kaynak kodunun* tamamını yazarak olsun , aynı temel adımları gerektirir.

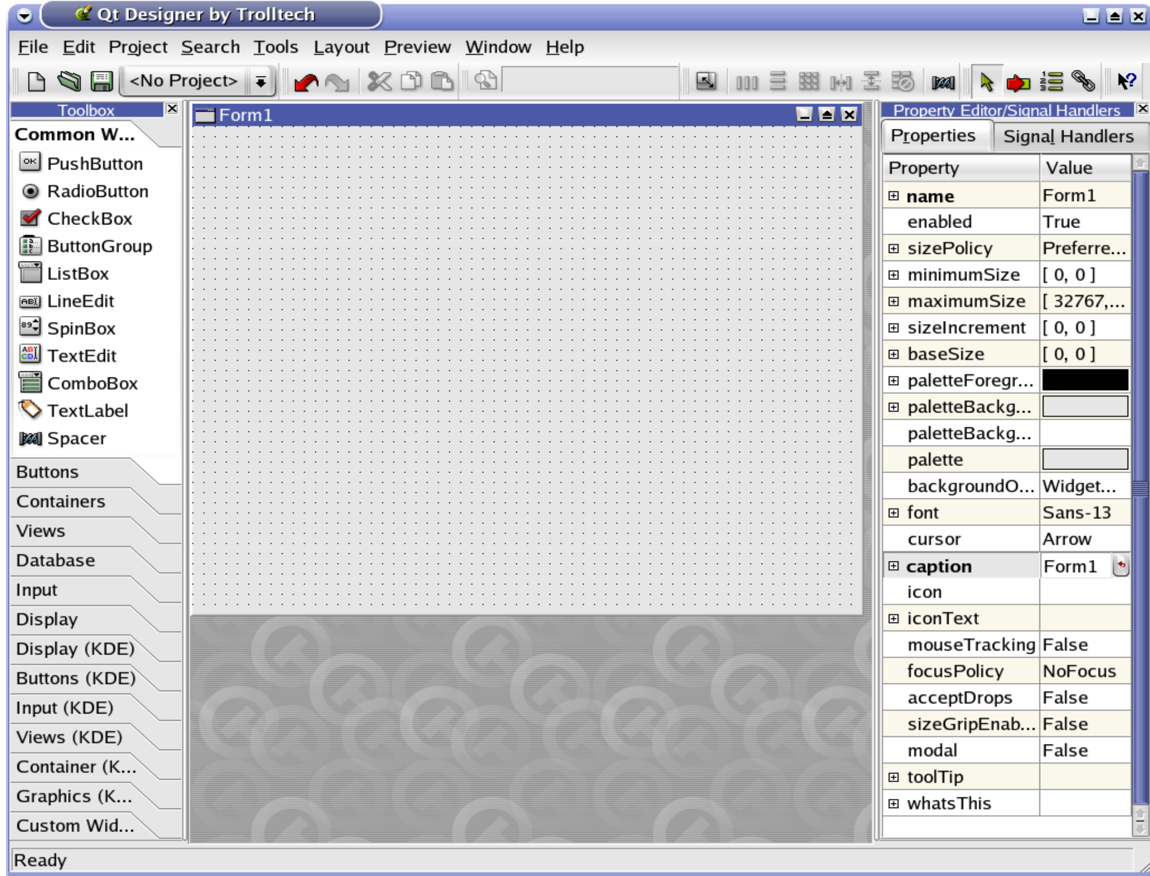
- Çocuk widgetları oluştur ve onların ilk ayarlarını yap.
- Çocuk widgetları *dizimlere* yerleştir.
- *Tab sırasını* ayarla.
- *Sinyal* ve *dilim* başlantılarını gerçekleştir.
- Diyaloğa mahsus *dilimleri* oluştur.

[11] benimAylığım
[12] aylık, maaş
[13] aylığıAyarla
[14] aylıkDeğişti



Şekil 2.4: Hücreye git diyaşoğu

Windows altında Qt designer proğramını başlatmak için Başla menüsünün altında Qt 3.2.x | Qt Designer a tıklayınız. Unix altında komut satırında `designer` yazmak ve Mac OS X altında Bulucu (Finder) da `designer` a iki defa tıklamak suretiyle *Qt designer* proğramını başlatabilirsiniz. Qt Desiğner başlatıldığında bir dizi *şablonun* bulunduğu bır diyalog ortaya çıkar. Diyalog (Dialog) şablonuna tıklayıp sonra OK düğmesine tıkladığınızda Form1 isminde bir pencere açılır.

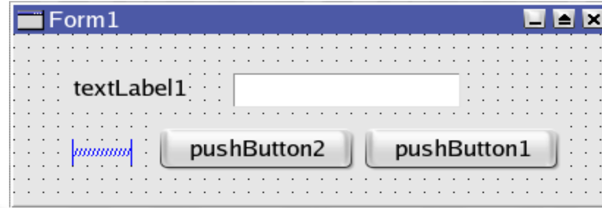


Şekil 2.5: Qt desiğner ve bir boş form.

İlk adım çocuk aletleri (widgets) oluşturup onları for üzerine yerleştirmektir. Bir tane etiket (text label), bir tane satır editörü (line editor), bir tane (yatay) *aralıkçı* (spacer) ve iki tanede

düğme (push buttons) oluşturunuz. Bunların her biri için ekranın sol tarafındaki alet kutusundakı ismine tıklayıp sonrada form üzerindeki yerine tıklayınız. Şimdi formun alt kısmını fareyi kullanarak yukarı doğru çekiniz. Sonuçta şekil 2.6 da izhar edilen bir şekil elde etmiş olmalısınız. Her bir ferdi form üzerinde yerli yerine yerleştirmek için fazla zaman harcamayınız. Birazdan Qt nin dizim mekanizmasini kullanarak istediğimiz şekilde her bir ferdi yerleğtireceğiz.

Aralıkçı (spacer) Qt designer içerisinde mavi yay şeklinde tezahur eder ancak proğram çalışırken gizlenir.



Şekil 2.6: Bir kaç alet içeren bir form.

Qt designerin ana penceresi içerisindeki *özellik editörünü* (property editor) kullanarak her bir aletin özelliklrini ayarlayınız:

1. Etikete (text label) tıkla. Onun ismini (`name`) "label" (etiket) olarak değiştiriniz ve metnini (`text`) "&Hücre Pozisyonu":
2. Satır editörüne (line editor) tıkla. Onun ismini (`name`) "lineEdit" olarak değiştir.
3. Aralıkçıya (spacer) tıkla. Onun yön (`orientation`) özelliğininin yatay ("Horizontal") olarak ayarlanmış olduğuna dikkat et.
4. İkinci düğmeye tıkla. Onun ismini (`name`) "okButton" olarak değiştir ve muktedir (`enabled`) özelliğini menfi (`False`), varsayılan (`default`) özelliğini müsbet (`True`) ve metin (`text`) özelliğini "Git" olarak değiştir.
5. İlk düğmeye tıkla. Onun ismini (`name`) "cancelButton" ve metnini (`text`) "Vazgeç" olarak değiştir.
6. Formun ardalanına tıklayıp onun ismini (`name`) "GoToCellDialog" ve başlığını (`caption`) "Hücreye Git" olarak değiştir.

Etiket haricinde, ki o *&Hücre Pozisyonu* göstermektedir, bütün aletler hazır vaziyetteler. *Tools|Set Buddy* menüsüne tıkladıktan sonra etikete (text label) tıklayınız ve lastiği satır editörüne kadar çekip serbest bırakınız. Şimdi etiketin *Hücre Pozisyonu* olrak teşhir edildiğini ve *ahbabınında* satır editörü (line editör) olduğunu göreceksiniz. Bunu etiketin ahbab (`buddy`) özelliğinin "lineEdit" olarak ayarlanmış olup olmadığına dikkat ederek tahkik edebilirsiniz.



Şekil 2.7: Yukarıdakı formun özellikleri ayarlanmış hali.

Şimdi yapılması gereken iş aletleri form üzerine dizimleri kullanarak yerleştirmek.

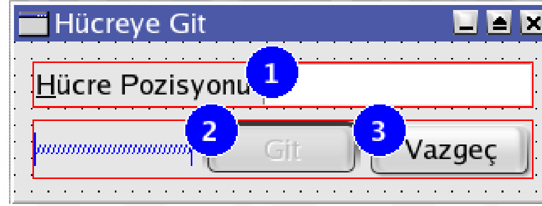
1. `Hicre Pozisyonu`na tıkla ve *Shift* tuşunu basılı tutarak yanı başındakı metin editörüne (line edit) tıkla böylece bunların her ikisi seçilmiş olur. Ana menüden *Layout*|*Lay Out Horizontally* komutunu seç.
2. Aralıkçıya tıkla ve *Shift* tuşu basılı olduğu halde sağ tarafta kalan *Git* ve *Vazgeç* düğmelerine tıkla taki bunların üçü birden seçilmiş olsun. Ana menüden *Layout*| *Lay Out Horizontally* komutunu seç.
3. Formun ardalanına tıkla bu sayede bütün seçimler iptal olmuş olur. Sonra ana menüden *Layout*|*Lay Out Vertically* komutunu seç..
4. Ana menüden *Layout*|*Adjust Size* komutunu kullanarak formu en ideal büyülüğüne getiriğniz.

Form üzerinde görünen kırmızı çizgiler dizimleri göstermektedirler ancak proğram çalışırken bunlar görünmezler.



Şekil 2.8: Form *dizimler* ile beraber gösterilmektedir.

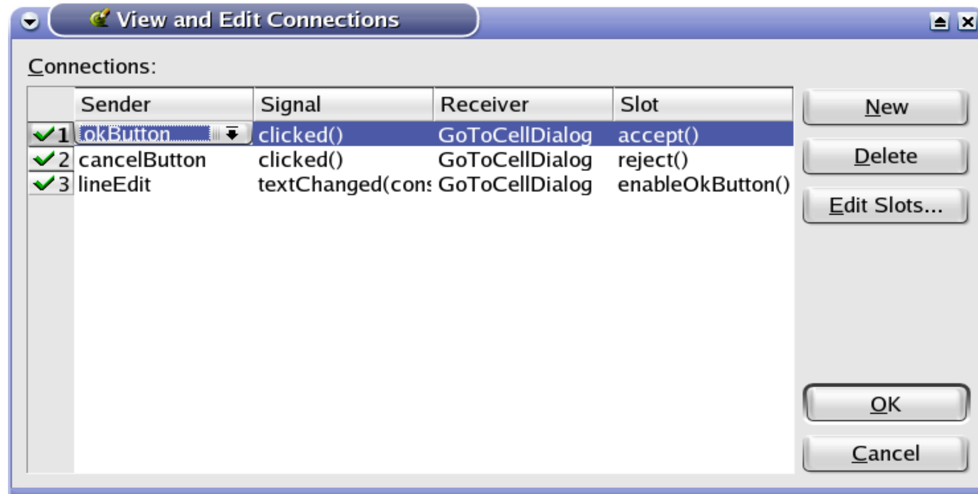
Şimdi yine ana menüden *Tools*|*Tab Order* komutunu seçiniz. *Odak* kabul edebilen her bir aletin yanı başında mavi daire içerisinse bir rakamın tezahür ettiğini göreceksin. *Tab sırasını* deüiştirmek istiyorsanız aletlerin her birine istediğiniz sıraya gire tıklayınız ve sonrada *Esc* tuşuna basınız.



Şekil 2.9: Formun *tab sırası* ayarı.

Formun tertibini bitirdik şimdi artık *sinyal-dilim* bağlantıları kurup bir takım hususi dilimler oluşturabiliriz. *Bağlantı editörünü* (connection editor) görüntülemek için ana menüden *Edit*|*Connections* komutuna tıklayınız (şekil 2.10).

Üç tane bağlantı kurmamız gerekiyor. Bağlantı yapmak için *New* düğmesine tıkal ve gönderici (*Sender*), sinyale (*Signal*), alıcı (*Receiver*) ve dilimi (*Slot*) açılan bileşim kutularını kullanarak ayarla. `okButton` düğmesinin `clicked()`[15] sinyalini `GoToCellDialog` formunun `accept()`[16] dilimine bağla. `cancelButton` düğmesinin `clicked()` sinyalini `GoToCellDialog` formunun `reject()`[17] dilimine bağla. *Edit*|*Slots* menüsüne tıklayıp *Qt Designer* ın *dilim editörünü* (şekil 2.11) açınız ve `enableOkButton()`[18] ismiyle şahsi `(private)` bir dilim `(slot)` oluşturunuz. Şimdi satır editörünün (*line edit*) `textChanged(const QString &)`[19] sinyalini `GoToCellDialog` formunun `enableOkButton()` dilimine rapededиniz.



Şekil 2.10: *Qt Designer* in bağlantı editöründe (connection editor) *sinyal* ve *dilimlerin*

---

15 fare ile tıklandı

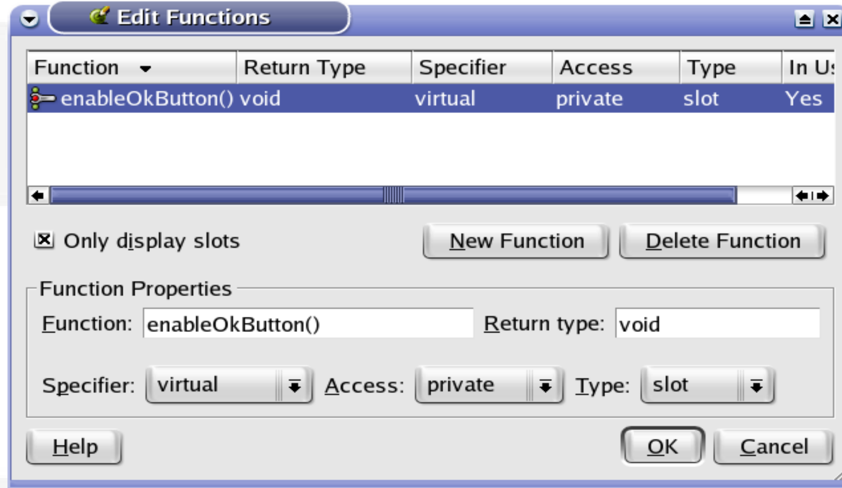
16 kabul et.

17 reddet.

18 okButton düğmesini (bizim için Git düğmesi) muktedir kıl.

19 metin degişti manasındakı bu sinyal satır editöründeki metin değişince yayınlanır.

rapdedilmiş hali



Şekil 2.11: *Qt Designer* in dilim editörü (slot editor)

Diyaloğa *Preview* | *Preview Form* menü seçeneğini kullanarak göz atabilirsiniz. Tab tuşuna bir kaç defa basmak suretiyle tab sırasını kontrol ediniz. *Alt+H* tuşuna basmak suretiyle satır editörünü odak noktası yapabilirsiniz. Diyaloğu kapatmak için *Vazgeç* düğmesine basınız.

Diyaloğu gotocelldialog.ui ismi ile gotocell dizini altına kaydediniz ve yine aynı dizin altında main.cpp ismi ile her hangi bir metin editörü kullanarak bir dosya oluşturun:

```
#include <qapplication.h>
#include "gotocelldialog.h"
int main(int argc, char *argv[])
{
      QApplication app(argc, argv);
      GoToCellDialog *dialog = new GoToCellDialog;
      app.setMainWidget(dialog);
      dialog->show();
      return app.exec();
}
```

Şimdi *qmake* proğramını kullanarak *.pro* ve *Makefile* dosyalarının oluşturunuz (*qmake -project; qmake gotocell.pro*). *qmake* proğramı kullanıcı arabirim dosyası olan *gotocelldialog.ui* dosyasının varlığını farkeder ve *gotocelldialog.h* ve *gotocelldialog.cpp* dosyalarını oluşturmak için gerekli kuralları *Makefile* dosyası içerisisne yerleştirir. Kullanıcı arabirim dosyası olan *.ui* dosyası, Qt nin kullanıcı arabirim derleyicisi olan *uic* proğramı tarafından C++ diline çevrilir.

Qt Designer proğramının en güzel yönlerinden birisi kaynak koduna dokunmaksızın form tasarımını değiştirmeye izin vermesidir. Qt Desiğner kullanmadan for tasarımını tamamen kaynak kodunu yazarak yaparsanız sonradan tasarım değişikliği yapmaya kalktıpınızda

bunun çok zaman alıcı bir amel oldupunu göreceksiniz. Bilakis Qt Designer kullanarak for tasarladığınızda uic proğramı otomatik olarak formları C++ diline çevirdiği için tasarımda yapılan herhangi bir tağayyur zaman kaybına neden olmaz.

Proğramı şimdi koşturabilirsiniz ancak onun istediğimiz şekilde davranmadığına müşahede edeceksiniz:

- Git düğmesi malul (disabled).
- Satır editörüne herhagi bir metin girilebilir, halbuki sadece geçerli sel adresleri kabul etmelidir.

Bu problemleri çözmek işin biraz kod yazmamız gerekli. Formun araalanına iki defe tıklamak suretiyle Qt nin kod editörünü açınız. Editöre şu kodu giriniz:

```
#include <qvalidator.h>
void GoToCellDialog::init()
{
      QRegExp regExp("[A-Za-z][1-9][0-9]{0,2}");
      lineEdit->setValidator(new QRegExpValidator(regExp, this));
}
void GoToCellDialog::enableOkButton()
{
      okButton->setEnabled(lineEdit->hasAcceptableInput());
}
```

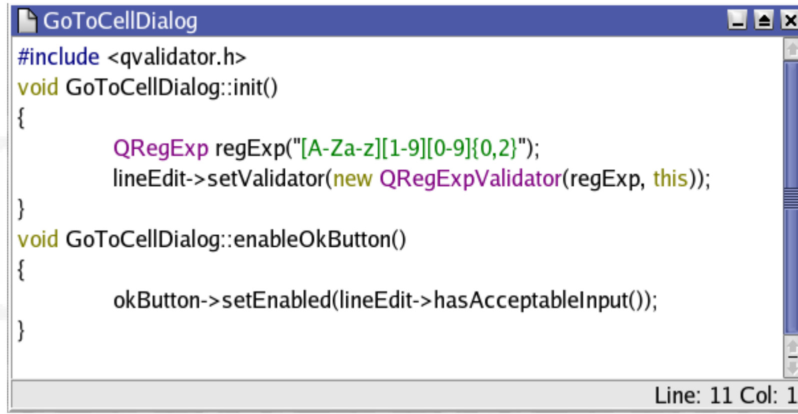
`init()` fonksiyonu otomatik olarak *uic* proğramı tarafından meydana getirilen *yapıcının* (constructor) sonunda çağrılır. *Satır editörünce* makbul sayılan girdileri sınırlamak için bir *müteberperver* (müteberperdaz ,validator) kullanacağız. Qt üç tane *müteberperver* sınıf tedarik eder: `QIntValidator`[20], `QDoubleValidator`[21], ve `QRegExpValidator`[22]. Burada biz *muntazam ibaresi* "[A-Za-z][1-9][0-9]{0,2}" olan bır `QRegExpValidator` kulanıyoruz ki bu ıbare şu anlama gelir: Büyük veya küçük bir harf, ardından 1 ila 9 arasında bir sayı ve bunu takiben iki veya daha az mütaakip 0 ile 9 arasında bir sayı. Qt de *muntazam ibareler* için başlıbaşına QRegExp isminde bir sınıf mevcuttur.

`this` timsalini `QRegExpValidator` yapıcısında kullanmak suretiyle onu `GoToCellDialog` *nesnesinin* çocuğu yaptık. Binaenaleyh daha sonra `QRegExpValidator` ün silinip silinmemesi konusunda kafa yormamıza gerek yok çünkü Qt otomatik olarak atası silinen (*yetim*) bütün çocukları siler. Satır editörünün *müteberperver* nezdinede makul bir metin içerip içermediğine bağlı olarak `enableOkButton()` dilimi *Git* düğmesini ya *muktedir* yada *malul* kılar. `QLineEdit::hasAcceptableInput()` fonksiyonu `init()` içerisinde *satır editörü* ile alakadar yaptığımız *müteberperverı* kullanır.

[20] Sadece tam sayıları (integer) kabul eder

[21] Sadece real sayıları (double, float) kabul eder

[22] Muntazam ibareler en genel anlamda bır müteberperverdir.



Şekil 2.12: Qt Designer ın kod editörü.

Yukarıda gösterilen kaynak kodunu yazdıktan sonra kydet. Bu kayıt iki dosyayı etkiler: kullanıcı arabirim dosyası olan gotocelldialog.ui dosyası ile C++ kaynak kodu ihtiva eden gotocelldialog. ui.h. Make komutunu kullanarak proğramı yeniden derle. Satır editörüne A12 yazdığında Git düğmesinin muktedir olacağına mülahaza et. Sonra satır edştörüne gelişi güzel metin girmeye çalış ve *müteberperverin* nasıl sadece geçerli metnin yazılmasına izin verdiğine dikkat et. Vazgç düğmesine basarak diyaşüğu terkedebilirsin.

Bu alıştırmada diyaloğu Qt Designer kullanarak tertip tağyir ettik ve sonra *Qt Designer* in kod editürünü kullanarak kaynak kodu yazdık. Diyaloğun kullanıcı arabirimi `.ui` uzantılı XML formatında bir dosyaya ve kaynak kodu `.ui.h` (C++ kaynak kod dosyası) uzantılı bır dosyaya kaydedilir. Bu ayrım `.ui.h` dosyasını herhangi bir metin editörü kullanarak değiştirmek isteyenler için kolaylık sağlar.

Birde `.ui.h` yaklaşımına alternatif bir başka yaklaşımda *Qt Designer* kullanarak `.ui` dosyası oluşturup sonra `uic` tarafından husule getirilen sınıfın *varisi* olan bir *alt sınıf* meydana getirip bu sınıfa istenilen sinyaller, dilimler ve diğer başka ilaveler yapılabilir. Mesela, *Hücreye Git* diyaloğu için bu `GoToCellDialog` diyaloüğunun *alt sınıfı* olan `GoToCellDialogImpl` isminde bir sınıf oluşturup lüzumlu ilaveleri yapmaktan ibarettir. Ayrıca `.ui.h` dosyalarının bu yaklağımla kullanılması gayet basit olup nihai *başlık dosyası* bizim diyaloğumuz için şu şekilde olabilir:

```
#ifndef GOTOCELLDIALOGIMPL_H
#define GOTOCELLDIALOGIMPL_H
#include "gotocelldialog.h"
class GoToCellDialogImpl : public GoToCellDialog
{
      Q_OBJECT
public:
      GoToCellDialogImpl(QWidget *parent = 0, const
                          char *name = 0);
private slots:
      void enableOkButton();
};
#endif
```

`And this source  le:`

```
#include <qlineedit.h>
#include <qpushbutton.h>
#include <qvalidator.h>
#include "gotocelldialogimpl.h"
GoToCellDialogImpl::GoToCellDialogImpl(QWidget *parent,
                                        const char *name)
      : GoToCellDialog(parent, name)
{
      QRegExp regExp("[A-Za-z][1-9][0-9]{0,2}");
      lineEdit->setValidator(new QRegExpValidator(regExp, this));
}
void GoToCellDialogImpl::enableOkButton()
{
      okButton->setEnabled(lineEdit->hasAcceptableInput());
}
```

Bu yaklaşımı (alt sınıf oluşturma) tercih eden proğramcılar muhtemelen *temel* (base) sınıfa `GoToCellDialogBase` ismini verip *alt sınıfa*, ki o ana özellikleri ihtiva eder, `GoToCellDialog` ismini verir.

Uic proğramı Qt Designer ile tasarlanmış sınıflardan alt sınıf oluşturulmasını kolaylaştıracak bir takım komut satırı seçeneklerine tedarik eder. Başlık dosyası iskeleti oluşturmak için -subdecl seçeneğini ve mukabil kaynak kodu dosyası iskeleti oluşturmak için ise -subimpl seçeneğini kullanın. Bu kitapta biz .ui.h yaklaşımını tercih edeceğiz çünkü bu method en mürecceh olanıdır ve .ui.h dosyalarını alt sınıf dosyalarına çevirmek gayet kolaydır. Qt Designer kullanım klavuzunda “Designer Yaklaşımı[23]” adında bir bölüm mevcut olup alt sınıf yaklaşımı ile .ui.h yaklaşımı arasındaki teknik farklar hakkında burada bilgi bulunabilir. Bir başka bölüm (Diyalogların Oluşturulması) Qt Designer daki üyeler (Members) bölümünü kullnarak uic tarafından husule getirilen kodda nasıl üye değişkenlerin oluşturulabileceğini izah etmektedir.

Şekil Değiştiren Diyaloglar

Şu ana kadar karşılaştığımız diyaloglar nerede ve nasıl kullanılırlarsa kullnılsınlar hep aynı aletleri ihtiva eder. Bazı durumlarda şekil degiştiren yada değişik görünümler arzeden diyaloglar arzu edilir. Teğayyur edebilen diyaloglar arasında en yaygın olan iki türden biri *açılabilen* diyaloglar diğeri ise *çok sayfalı* diyaloglar. Her iki diyalog *Qt Designer* içerisinde tasarlanabileceği gibi kaynak kod yazmak suretiylede husule getirilenilir.

Açılabilen diyaloglar genelde basit görünüme sahip olup mevcut bir düğme vasıtası ile açik ve kapalı görünümü arasında seçim yapılabilir. *Açılabilen* diyaloglar yeni kullanıcılar için basit olsun diye detayları saklamaktadırlar ve tecrübeli kullanıcılar diyaloğu açıp detaylara inebilirler. Bu bölümde *Qt Designer* kullanarak açılabilen diyaloglar oluşturacağız (şekil 2.13).

---

[23] Malesef Qt Designer kullan m k lavuzu bu kitap kaleme al n rkan sadece ngilizce olarak mevcut idi.

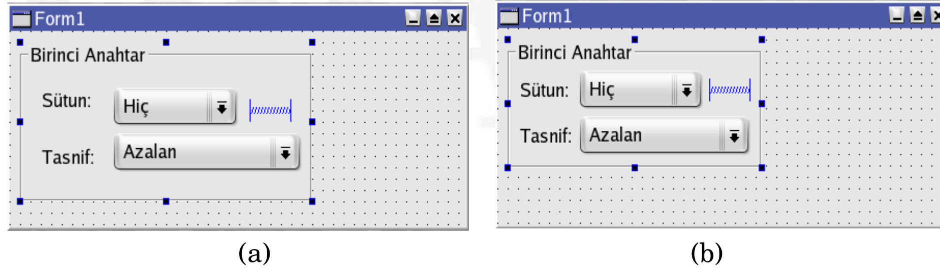


Şekil 2.13: Basit ve açılmış hali bir *tasnif* (sort) diyaloğu

Bu diyalog bir *spreadsheet* proğramında istimali muhtemel bir *tasnif* diyaloğudur ki kullanıcı bu diyaloğu kullanarak bir veya birden fazla sütunu sıraya sokabilir. Diyalog *basit* haliyle kullanıcıya bir tek *tasnif anahtarı* tedarik ederken *açılmış* hali ise iki *ilave anahtar* kullanıcıya sunar. Bir *Aç* düğmesi kullanıcının *basit* ve *açık* görüntüleri arasında değiştirebilir. *Qt Designer* da diyaloğu açılmış haliyle oluşturacağız ve ziyade olan ikinci ve üçüncü anahtarları grektiğinde proğram koşturulurken gizleyeceğiz. Alet her na kadar karışık gözüksede *Qt Designer* altında kolayca yapılabilir. İşin sırrı önce birinci anahtarı oluşturup daha sonra onu kopyalayarak ikinci ve üçüncü anahtarları oluşturmaktır:

1. Bir *grup kutusu* (group box), iki tane *etiket* (text label), iki tane *açılan kutu* comboboxes) ve bir tane yatay aralıkçı (spacer.)
2. Gurup kutusunun sağ alt köşesinden fare ile tutup büyütünüz.
3. Diğer aletleri gurp kutusuna naklet ve şekil 2.14 teşhir edildiği gibi yerleştirmeye çalış.
4. İkinci *açılan kutunun* (combobox) sağ kenarından tutup çek taki o ilk *açılan kutunun* (combobox) iki katı genişliğinde olsun.
5. Grup kutusunun başlığını (`title`) "&Birinci Anahtar", ilk etiketin `text` (metin) özelliğini "Sütun" ve ikinci etiketin `text` özelliğini "Tasnif" olarak değiştir:
6. İlk *açılan kutuya* (combobox) iki defa tıklayıp *Qt Desiğner* in *liste kutusu editörünü* (list box editor) açınız. Liste içerisisnde metni "Hiç" olan bir ferd (item) oluşturunuz.
7. İkinci *açılan kutuya* (combobox) iki defa tıklayıp "Azalan" ve "Artan" namında iki tane *ferd* oluşturunuz.

8. Gurup kutusuna tıkladıktan sonra *Layout|Lay Out in a Grid* menüsünü seçiniz. Böylece şekil 2.14 tekine yakın bir görüntü elde etmiş olursunuz.



(a) (b)

Şekil 2.14: Klavuz (ızgara) dizimi kulanılmadan öncce (a) ve sonra (b)

Şayet *grup kutusunun* görünümü istediğiniz şekli almadıysa *Edit|Undo* menüsüne tıklayıp aletleri şekil 2.14 (a) daki gibi düzenleyip sonra *Layout|Lay Out in a Grid* menüsünü tekrar seçiniz.

Şimdi ikinci ve üçüncü *anahtar kutularını* ekleyeceğiz:

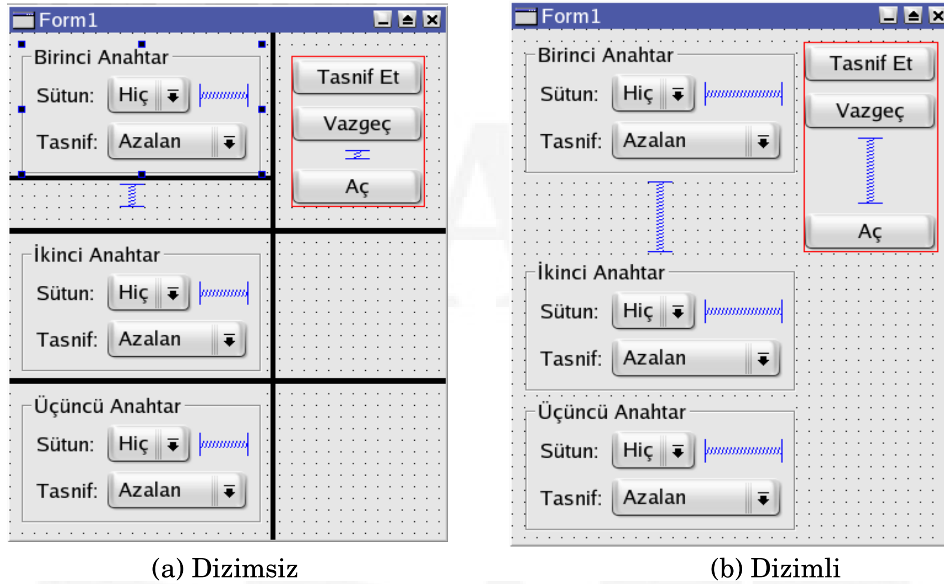
1. Formun yüksekliğini ayarla taki ilave aletler için yer olsun. Grup kutusunu seç sonra onu kopyala (*Edit|Copy*) ve *Edit|Paste* menüsünü kullanarak iki defa teksir et. Sonra her bir *grup kutusunu* onun nihai pozisyonuna yakın bir yere yerleştir.
2. Daha yeni çoğalttığın grup kutularının başlıklarını (`title`) "&İkinci Anahtar" ve "&Üçüncü Anahtar" olarak değiştir.
3. "Tasnif Et", "Vazgeç" ve "Aç" düğmelerini oluştur (bkz. şekil 2.15.)
4. "Tasnif Et" düğmesinin varsayılan (`default`), sakin (`toggle`) özelliklerini müsbet (`True`) yap.
5. İki tane düşey *aralıkçı* (spacer) oluştur.
6. *Tasnif Et*, *Vazgeç* ve *Aç* düğmelerini son ikisinin arasında bir düşey aralıkçı yerleştirmek şartiyle tertib et. Sonra bütün dört ferdi seç ve *Layout|Lay Out Vertically* menüsünü kullanarak onları bir düşey dizim içerisinde tanzim et.
7. İkinci aralıkçıyı birinci ve ikinci grup kutuları arasına yerleştir.
8. Her iki aralıkçının `sizeHint`[24] özelliğini (20, 10) olarak değiştir.
9. Aletleri şekil 2.15 (a) teşhir edildiği şekilde ızgara biçiminde düzenle.
10. *Layout|Lay Out in a Grid* menüsünü seçtiğinizde form şekil 2.15(b) de izhar edilen form ile muvafık olmalıdır.

Nihai ızgara dizimi dört sıra ve iki sütundan ibaret olup toplam sekiz hücreye sahiptir.

[24] İma edilen yada tavsiye edilen ebat

Birinci anahtar grup kutusu, en soldaki düşey aralıkçı, *ikinci anahtar* grup kutusu ve *üçüüncü anahtar* kutusu grup kutularından herbiri birer hücre istila etmektedirler. *Tasnif Et*, *Vazgeç* ve *Aç* düğmelerini ihtiva eden düşey dizim iki tane hücre işgal etmektedir. Geriye sağ alt tarafta iki tane hali hücre kaldı. Şayet sizin formunuz bi şekli almadıysa ızgara dizimini bozup yeniden deneyiniz.

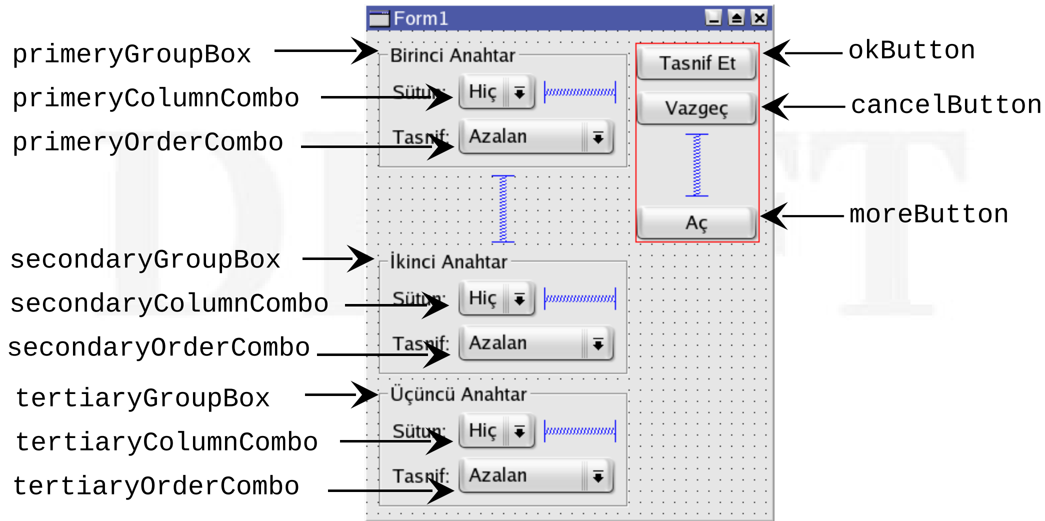
Formun ebat değiştirme tarzı (`resizeMode`) özelliğini “Auto” yerine “Fixed” yap ki o kullanıcının diyaloğun büyüklüğünü değiştiremsine mani olur. Böylece diyaloğun ebat ayarı, onun çocuklarından hangisinin zuhur edüp etmediğine bağlı olarak, tamamen dizim tarafından yapılır. Buda diyaloğun hem açık hemde kapalı halinde en müsait büyüklükte tezahur etmesini teminat altına alir.



(a) Dizimsiz (b) Dizimli

Şekil 2.15: Formun çcocuklarının izgara biçiminde dizilmiö haller.

Formun ismini (`name`) “SortDialog” ve başlığını (`caption`) “Tasnif” olarak değiştir. Çocuklarin isimlerini (`name`) ise şekil 2.16 da gösterilen isimlere değiştir.



Şekil 2.16: Formun ihtiva ettiği aletlerin isimleri

Şimdi artık bağlantıları kurabiliriz:

1. `okButton` düğmesinin `clicked()` sinyalini formun (`SortDialog`) `accept()` dilimine bağla.
2. `cancelButton` düğmesinin `clicked()` sinyalini formun (`SortDialog`) `reject()` dilimine rapt et.
3. `moreButton` düğmesinin `toggled(bool)` sinyalini `secondaryGroupBox` gurup kutusunun `setShown(bool)` dilimine kayd et.
4. `moreButton` düğmesinin `toggled(bool)` sinyalini `tertiaryGroupBox` grup kutusunun `setShown(bool)` sinyaline bend et.

Forma iki defa tıklamak suretiyle Qt Designerin C++ kod editörünü aç ve aşağıdaki kodu yaz:

```
void SortDialog::init()
{
   secondaryGroupBox->hide();
   tertiaryGroupBox->hide();
   setColumnRange( `A` , `Z` );
}
void SortDialog::setColumnRange(QChar first, QChar last)
{
   primaryColumnCombo->clear();
   secondaryColumnCombo->clear();
   tertiaryColumnCombo->clear();
   secondaryColumnCombo->insertItem(tr("None"));
   tertiaryColumnCombo->insertItem(tr("None"));
   primaryColumnCombo->setMinimumSize(
   secondaryColumnCombo->sizeHint());
   QChar ch = first;
   while (ch <= last) {
```

```
18          primaryColumnCombo->insertItem(ch);
19          secondaryColumnCombo->insertItem(ch);
20          tertiaryColumnCombo->insertItem(ch);
21          ch = ch.unicode() + 1;
22    }
23 }
```

`init()` fonksiyonu diyaloğun ikinci ve üçüncğ anahtar grup kutularını saklar.

`setColumnRange()` dilimi, açılan kutuları *elektronik çizelegedeki* (spreadsheet) seçilmiş olan sütunlara bağlı olarak doldurmaktadır. İhtiyari olan ikinci ve üçüncü grup kutularındaki açılan kutuları "Hiç" ile dolduruyoruz. Her ne kadar bu dilimi *Qt Designer* ın dilim editörünü kullanarak oluşturmadıysakta *Qt Desiğner* bu dilimin varlığını keşfeder ve *uic* otomatik olarak `SortDialog` sınfını oluştururken gerekli fonksiyon tanımın oluşturacaktır.

Satır 14 ve 15 de mühim bir dizim ibaresi takdim edilmektedir. `QWidget::sizeHint()` fonksiyonu aletin *en müsait ebatını* verir ve dizim mekanızması buna mümkün mertebe sadık kalmaya çalışır. Bu bazan farklı türden aletlere veya içeriği muhtelif olan hemcins aletlere dizim sisteminin neden ayrı boyutlar verdiğine bir nebze ışık tutar. Bu, açılan kutular için şu anlama gelir: ikinci ve üçüncü açılan kutular *Hiç* içerirler ve yanlız bir harfe mutevi olan birinci açılan kutudan daha büyük yer kaplarlar. Bundan ictinab etmek için birinci açılan kutunun asgari boyutunu ikinci açılan kutunun *en müsait ebatını* eşdeğer yapıyoruz.

İşte bir deneme amaçlı main() fonksiyonu ki o diyaloğun sanki C den F e kadar olan sütünlar seçilmiş gibi davranmasını sağlar ve diyaloğu ekranda gösterir:

```
#include <qapplication.h>
#include "sortdialog.h" int main(int argc, char *argv[])
{
      QApplication app(argc, argv);
      SortDialog *dialog = new SortDialog;
      app.setMainWidget(dialog);
      dialog->setColumnRange( `C` ,  `F` );
      dialog->show();
      return app.exec();
}
```

Böylece açılabilen diyaloğu ikmal etmiş olduk. Bu misal nümayişkarane[25] gösteriyorki *açılan diyalogların* tasarımı ve işletimi *basit diyaloglardan* çok zor değildir: İhtiyaç duyulan ziyade şeyler bir *sakin* (toggle) düğmesi, bir kaç *sinyal* ve *dilimler* ile boyutları kullanıcı tarafından ayarlanamayan bir dizimden ibarettir. Sıkça kullanılan bir diğer diyalogda *çok sayfalı diyalogdur* ki onun tasarım ve işletimi açılan diyalogdan daha kolaydır. Bu tür diyaloglar bir kaç muhtelif şekilde oluşturulabilirler:

- Bir `QTabWidget` başlı başına kullanılabilir. Üst tarafta bir *sekme çubuğu* tedarik eder ki o Qt de mevcut alet yığını sınıfı olan QwidgetStack sınıfını idare eder.

---

[25] Tatbikat ile izah ve isbat edercesine

- Bir `QListBox` ve bir `QWidgetStack` birlikte kulanılır ki `QListBox` ın hali hazırda seçilmiş olan ferdi `QWidgetStack` yığınındakı aletlerden hangisinin ekrana taşınacağını belirler.
- Bir QListView veya bir QiconView, QListBox gibi QWidgetStack ile beraber kullanılabilir. QWidgetStack sınıfı altıncı bölümde Dizim Mekanızmalrı altında ele alınacaktır.

Mütaharrik Diyaloglar

Mütaharrik diyaloglar Qt Designer ın .ui dosyalarından proğram koşturulurken husule getirilen diyaloglardır. Mütaharrik diyaloglar uic tarafından C++ kodu çevrilmezler, bilakis proğram koşturulurken QWidgetFactory sınıfı vasıtasıyla şu şekilde yüklenirler:

```
QDialog *sortDialog = (QDialog *)
      QWidgetFactory::create("sortdialog.ui");
```

Formun çocuklarına `Qobject::child()` vasıtasıyla şu şekilde ulaşabiliriz:

```
QComboBox *primaryColumnCombo = (QComboBox *)
      sortDialog->child("primaryColumnCombo", "QComboBox");
```

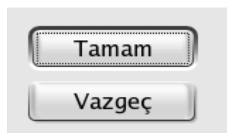
Şayet diyalog arnan isim ve türden bir çocuğa sahip değil ise `child()` fonksoyonu *boş bir timsal* (null pointer) getirir. `QWidgetFactory` sınıfı müstakil bir kütüphane içerisinde yer almaktadır. `QWidgetFactory` sınıfını bir Qt proğramı içerisinde kullanabilmek için şu satırı eklememiz gerekir:

```
LIBS += -lqui
```

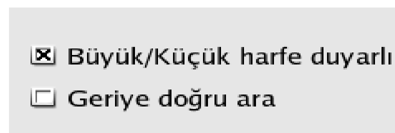
Bu *nahiv* (syntax) bütün işletim sistemleri altında çalışır her ne kadar Unix sisteminin nehvine benziyor isede. *Müteharrik diyaloglar,* proğramı yeniden derlemeden formun dizimini değiştirme imkanı sağlarlar. *Qt Desiğner* ın kullanım klavuzunda *Alt Sınıf Oluşturma ve Müteharrik diyaloglar* (Subclassing and Dynamic Dialogs) başlığı altında bu diyalogları kullanan kamil bir misal mevcuttur.

Mevcut Qt Aletleri ve Diyalogları

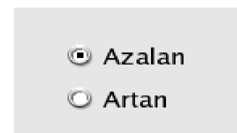
Qt, çoğu proğramlar için yeterli olabilecek alet ve diyaloglar tedarik etmektedir. Bu bölümde hemen hemen bunlarının bütününün ekran resimlerini teşhir edeceğiz. Bir kaç hususi aletler tehir edilecektir: `QMenuBar`, `QPopupMenu`, gibi `QToolBar` *ana pencere* aletleri üçüüncü bölümde ele alınacaktır ve `QDataView` ve `QDataTable` gibi veritabanı aletleri 12. bölümde tafsil edilecektir. Mevcut aletlerin çoğu bu kitapda takdim edilen misallerde istimal olunacaktır. Aşağıdaki ekran resimleri Linux işletim sistemi altinda alınmıştır.



QPushButton



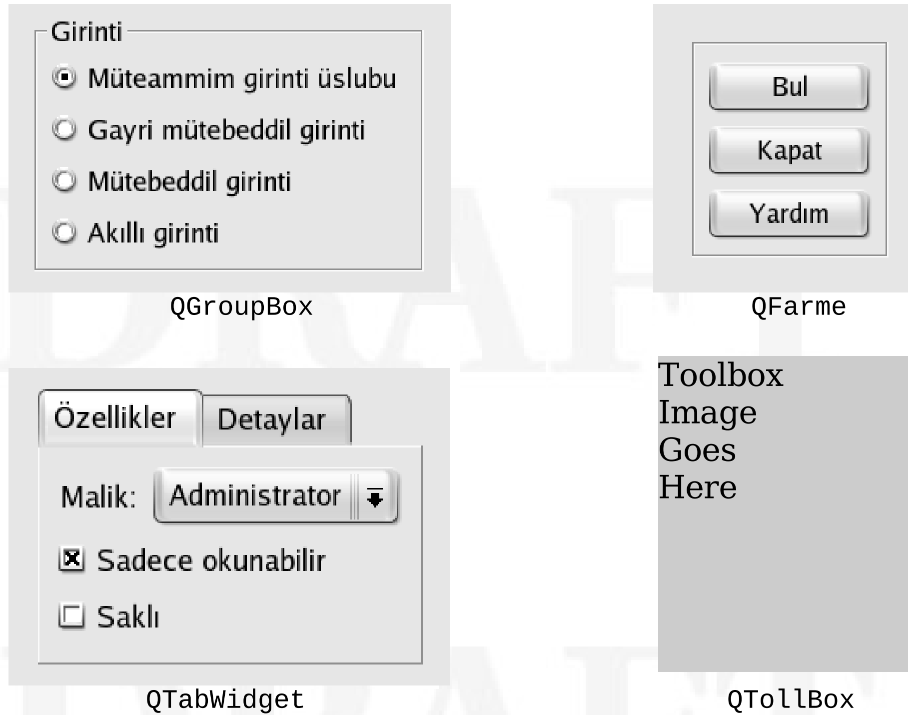
QcheckBox



QRadioButton

Şekil 2.17: Qt düğme aletleri.

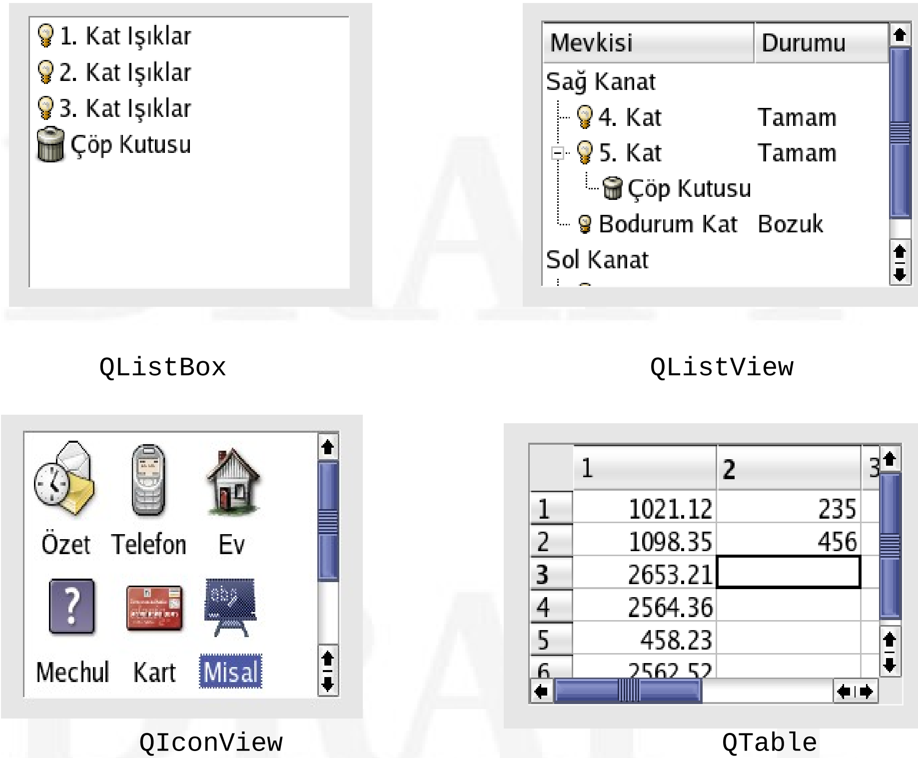
Qt üç çeşit "düğme" tedarik etmektedir: `QPushButton`, `QCheckBox`, and `QRadioButton`. `QPushButton` en yaygın olarak bir defa tıklandıklarında bir fiil yada işin husule getirilmesine neden olsun diye kullanılırlar. Bazanda *sakin düğme* (toggle button) şeklinde kullanılırlar. `QradioButton` düğmeleri genellikle `QButtonGroup` içerisinde kullanılırlar. Grup işerisindeki düğmelerden yanlız ve yanlız bir tanese seçilebilir. Bunu karşılık `QCheckBox` bağımsız açık/kapalı seçenekleri olarak kullanılabilir.



Şekil 2.17: Qt muhtevi (container) aletleri.

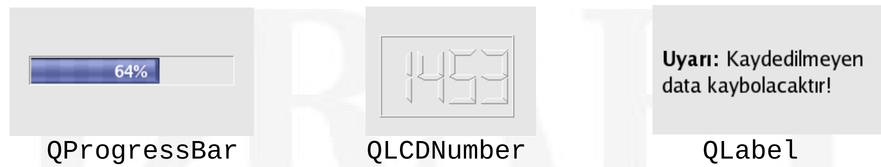
Qt nin *muhtevi aletleri* başka aletleri ihtiva eden yani içine alan aletlerdir. `QFrame` üzerine çizim yapılması maksadıyla yanlız başına kullanılabilir ve başta `QLabel` ve `QLineEdit` olmak üzere bir çok sınıflar onun varısidirler. `QButtonGroup`, `QGroupBox` sınıfına çok yakın olması nedeniyle, burada gösterilmedi.

`QTabWidget` ve `QToolBox` *çok sayfalı aletlerdir*. Bunların ihtiva ettiği her bir sayfa birer çocuk alettir ve sıfırdan başlayarak numaralandırılmışlardır.



Şekil 2.18: Qt nin *teşhir* (item views) aletleri.

Teşhir (item views) aletleri çok miktarda data ile en eşverişli çalışabilecek şekilde tasarlanmışlardır ve genellikle kaydırıcı çubuğu kullanırlar. Kaydırıcı çubukları `QScrollView` sınıfı tarafından tedarik edilmektedir ve , kı bu sınıfa diğer bir çok *teşhir* sınıfını varisdirler.



Şekil 2.19: Qt nin *teşhir* (display) aletleri.

Qt sadece bilgi teşhiri için kullanılabilecek bir kaç sınıf tedarik etmektedir. Bunlar arasında en mühimi QLabel sınıfıdır ve o rich text (HTML türü bir nehv kullanarak) ve resimleri görüntülemek maksadıyla kullanılabilir.

QTextBrowser[26] sınıfı QtextEdit sınıfının bir alt sınıfı olup HTML formatından listeler, tablolar, resimler hipertext linkleri ve benzerlerini görüntüleme yeteneğine sahipdir; Qt Assistant proğramı QTextBrowser kullanarak Qt kullanma klavuzunu gösterir.

---

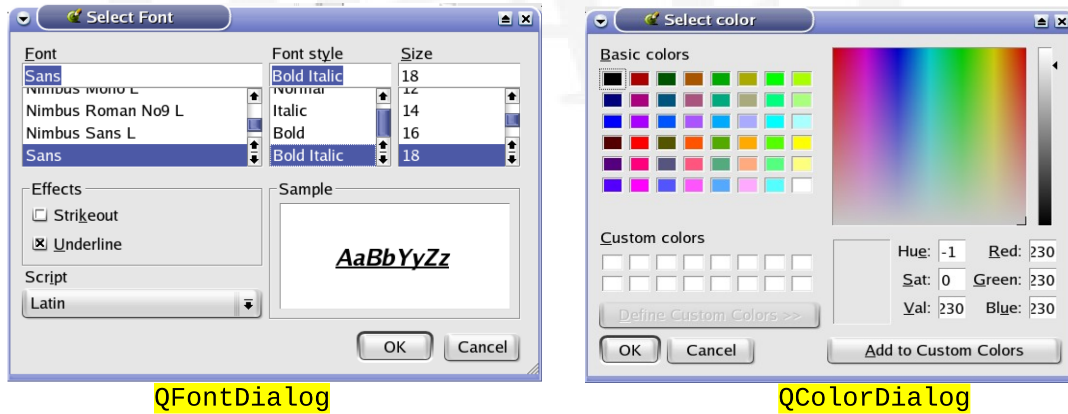
[26] Bu sınıf QtextEdit sınıfının aksine sadece metnin gürüntülenmesi maksadıyla kullanılır. Kullanıcının metni değiştirmesine izin vermez.



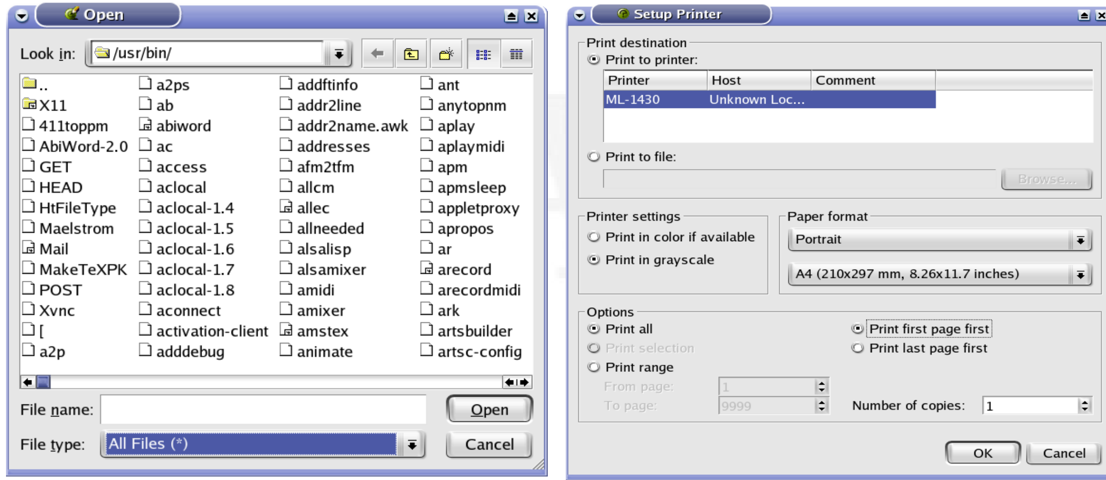
Şekil 2.20: Qt nin *girdi* (input) aletleri.

Data girmek için Qt bir çok alet tedarik eder. `QLineEdit` bir girdi maskesi yada muteberperver kullanmak suretiyle makbul girdisşnş mahdud eder. `QtextEdit`, `QscrollView` sınıfının bir alt sınıfı olup büyük miktadrda data ile çalışmaya kadirdir.

Qt, sıkça kullanılan font, renk seçme, dosya seçme ve yazıcıya gönderme diyaloglarını temin eder.

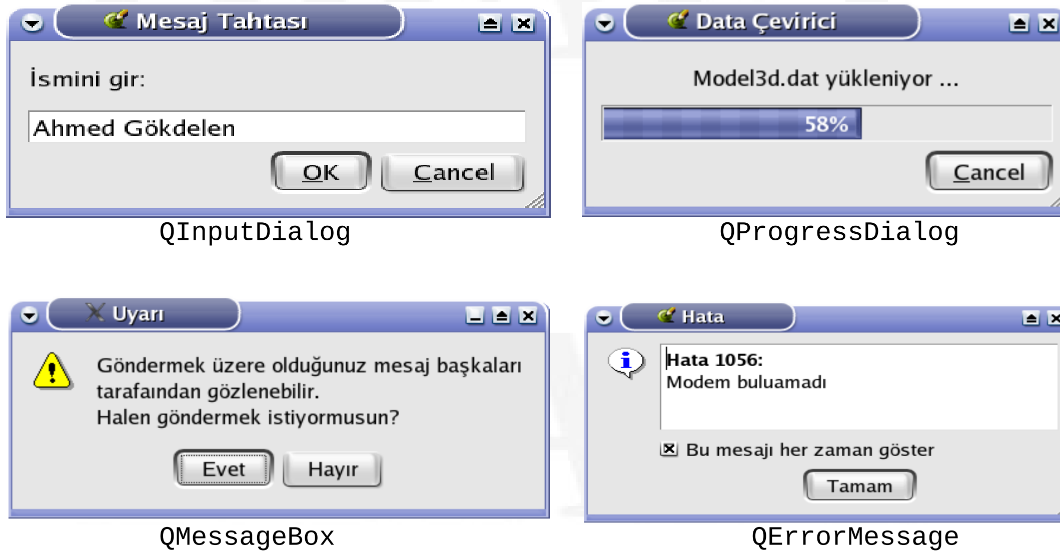


Şekil 2.21: Qt nin font ve renk diyalogları



Şekil 2.22: Qt nin dosya seçme ve yazıcıya gönderme diyalogları

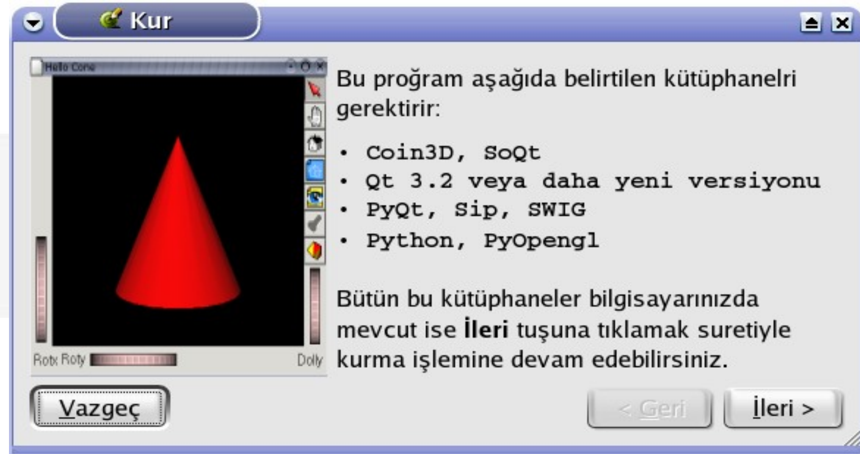
Windows Mac OS X işletim sistemleri altında Qt mümkün ise systemin kendisine mahsus diyalogları kullanmaktadır.



Şekil 2.22: Qt nin informasyon diyalogları

Qt çok amaçlı bir mesaj kutusu ve hata diyaloğu tedarik eder ki o izhar etmiş olduğu msaji hatırında tutuar. Zaman alıcı işlemlerin terakkisini yada ilerleyişini göstermek için ya `QprogressDialog` yada sabıkan gösterilen `QProgressBar` istimal edilebilir. Kullanıcıtan tek bır satırlık girdi yada bir tek rakam taleb etmek için `QinputDialog` gayet elverişlidir.

`QWizard` sınıfı yardımcı diyloglar (wizarda) oluşturmak için kullanılır. *Qt Designer* da yardımcı diyaloglar husule getirmek için bir *şablon* (Wizard template)tedarik etmektedir.



Şekil 2.22: Qt nin `QWizard` diyalogları

Qt kullanıma hazır çok sayıda alet ve diyalog tedarik etmektedir. Sinyal ve dilim mekanizmasını kullanarak ve ihtiyaca göre dilimler yazmak suretiyle bu aletler vi diyaloglar hususileştirilebilirler. Bazı durumlarda yep yeni bir alet oluşturma şhtşyacı doğabilir. Qt bu işi kolaylaştırmaktadır ve hususi aletler Qt nin kendi aletleri gibi işletimsisteminden bağımsız çizim fonksiyonlarını kullanabilirler. Hatta hususi aletler Qt Designer a ilave edilerek Qt nin kendi aletleri gibi kullanılabilirler. Beşinci bölümde hususi aletlerin nasıl yapıldığını göreceğiz.